# EmailClassiffier

June 25, 2024

## 0.1 Project: Email Spam Classifier


- Author: Naveed Ur Rehman
- Submitted to : Ezitech


## 0.2 Overview

This project focuses on building a classifier to distinguish between spam and non-spam emails. Various machine learning models are explored and evaluated for their accuracy in predicting email classifications.

### 0.2.1 Importing Libraries

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

## Convert data into vector
from sklearn.feature_extraction.text import CountVectorizer
## Data spliting into train and test set
from sklearn.model_selection import train_test_split

## Dealing with textul data
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

# nltk.download('punkt')
# nltk.download('stopwords')

## Models
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import GaussianNB,MultinomialNB
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import␣
↪accuracy_score,classification_report,confusion_matrix
```

### 0.2.2 Importing datasets

[ ]:
```
df = pd.read_csv("data/emails.csv")
df.head()
```

[ ]:
```
                                                text  spam
0  Subject: naturally irresistible your corporate…     1
1  Subject: the stock trading gunslinger  fanny i…     1
2  Subject: unbelievable new homes made easy  im …     1
3  Subject: 4 color printing special  request add…     1
4  Subject: do not have money , get software cds …     1
```

[ ]:
```
df1 = pd.read_csv("./data/spam_ham_dataset.csv")
df1.head()
```

[ ]:
```
   Unnamed: 0 label                                               text  \
0         605   ham  Subject: enron methanol ; meter # : 988291\r\n…
1        2349   ham  Subject: hpl nom for january 9 , 2001\r\n( see…
2        3624   ham  Subject: neon retreat\r\nho ho ho , we ' re ar…
3        4685  spam  Subject: photoshop , windows , office . cheap …
4        2030   ham  Subject: re : indian springs\r\nthis deal is t…

   label_num
0          0
1          0
2          0
3          1
4          0
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5728 entries, 0 to 5727
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    5728 non-null   object
 1   spam    5728 non-null   int64
dtypes: int64(1), object(1)
memory usage: 89.6+ KB
```

[ ]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 5171 entries, 0 to 5170
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  5171 non-null   int64
 1   label       5171 non-null   object
 2   text        5171 non-null   object
 3   label_num   5171 non-null   int64
dtypes: int64(2), object(2)
memory usage: 161.7+ KB
```

[ ]:
```python
df.shape,df1.shape
```

[ ]: ((5728, 2), (5171, 4))

### 0.2.3 Data cleaning

[ ]:
```python
df1.drop(['Unnamed: 0','label'],axis=1,inplace=True)
df1
```

[ ]:
```
                                                   text  label_num
0     Subject: enron methanol ; meter # : 988291\r\n…          0
1     Subject: hpl nom for january 9 , 2001\r\n( see…          0
2     Subject: neon retreat\r\nho ho ho , we ' re ar…          0
3     Subject: photoshop , windows , office . cheap …          1
4     Subject: re : indian springs\r\nthis deal is t…          0
…                                                   …        …
5166  Subject: put the 10 on the ft\r\nthe transport…          0
5167  Subject: 3 / 4 / 2000 and following noms\r\nhp…          0
5168  Subject: calpine daily gas nomination\r\n>\r\n…          0
5169  Subject: industrial worksheets for august 2000…          0
5170  Subject: important online banking alert\r\ndea…          1

[5171 rows x 2 columns]
```

[ ]:
```python
df1.rename({"label_num":"spam"},axis=1,inplace=True)
```

[ ]:
```python
sum(df.duplicated()),sum(df1.duplicated())
```

[ ]: (33, 178)

[ ]:
```python
df.shape,df1.shape
```

[ ]: ((5728, 2), (5171, 2))

[ ]:
```python
df.drop_duplicates(inplace=True)
df1.drop_duplicates(inplace=True)
```

[ ]:
```
df.shape,df1.shape
```

[ ]:
```
((5695, 2), (4993, 2))
```

### 0.2.4 Data Merging

[ ]:
```
combine_df = pd.concat([df,df1])
combine_df.head()
```

[ ]:

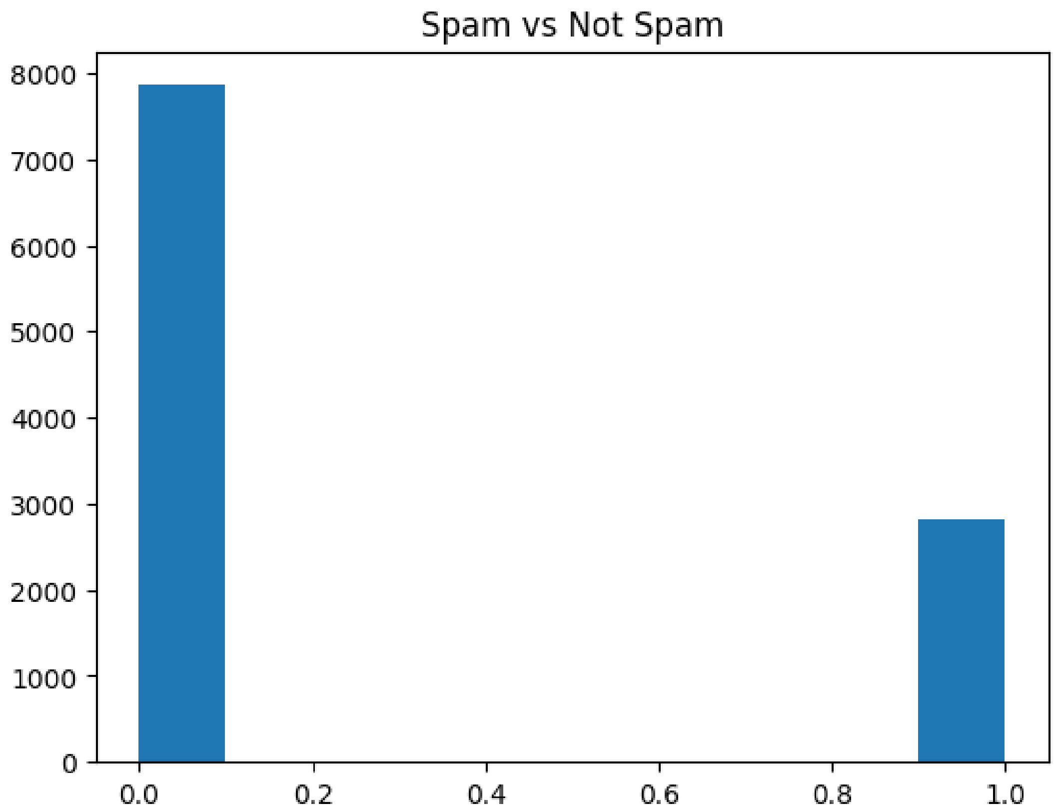
| | text | spam |
|---|---|---|
| 0 | Subject: naturally irresistible your corporate… | 1 |
| 1 | Subject: the stock trading gunslinger  fanny i… | 1 |
| 2 | Subject: unbelievable new homes made easy  im … | 1 |
| 3 | Subject: 4 color printing special  request add… | 1 |
| 4 | Subject: do not have money , get software cds … | 1 |

[ ]:
```
sum(combine_df.duplicated())
```

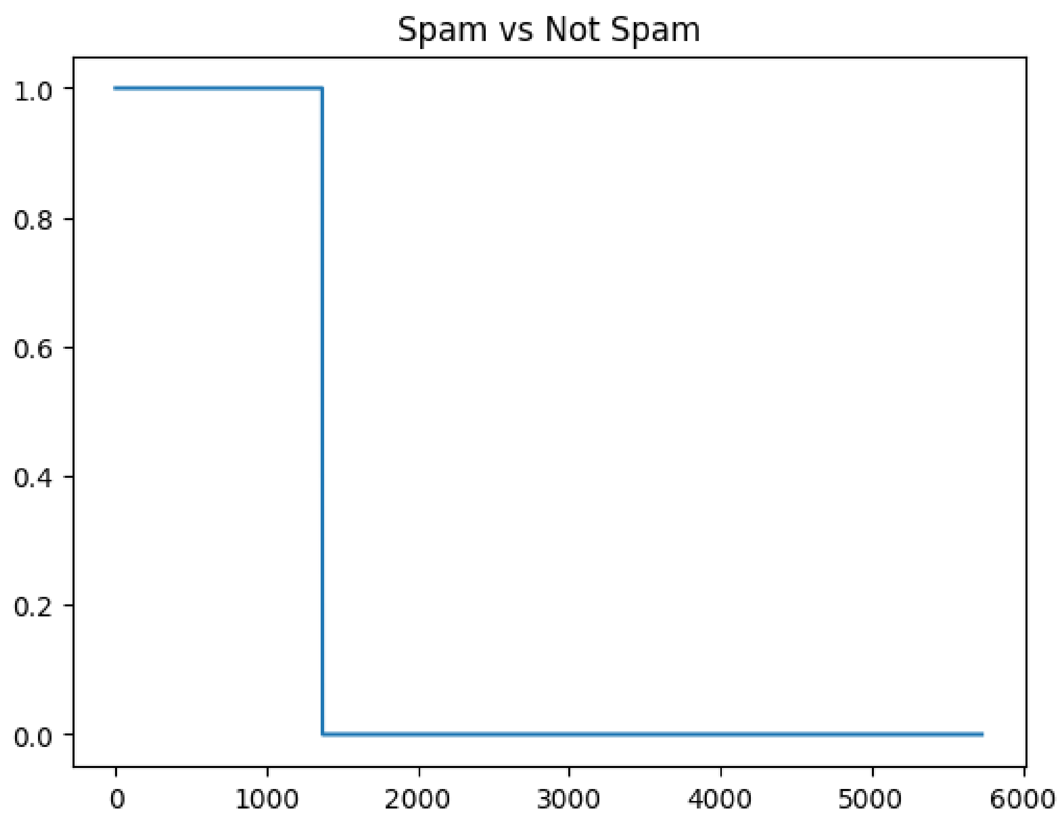
[ ]:
```
0
```

### 0.2.5 Data Visulaization

[ ]:
```
plt.hist(combine_df['spam'])
plt.title("Spam vs Not Spam")
plt.show()
```




[ ]:
```
df['spam'].plot.line()
plt.title("Spam vs Not Spam")
plt.show()
```




#### 0.2.6 Preprocessing

- Convert to lowercase
- Remove Unneccessary characters such as Punctution and Special Characters
- Remove stop words
- Perform stemming or lemmatization
- Handle missing values if any

- There is no missing data point in the dataset

[ ]:
```
# Remove Unneccessary characters such as Punctution and Special Characters
import re
import string

def remove_punc_spec_chars(text):
    # Remove punctuation
    text = text.translate(str.maketrans("","",string.punctuation))

    # Remove special characters (keeping alphanumeric characters and spaces)
    text = re.sub(r"[^a-zA-Z0-9\s]","",text)

    # Remove extra whitespace
```

```
    text = re.sub("\s+"," ",text).strip()
    return text
```

[ ]:
```
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove punctuation and special characters
    text = remove_punc_spec_chars(text)

    # Tokenize
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # Stemming
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]

    # Join tokens back into a string
    preprocessed_text = ' '.join(tokens)

    return preprocessed_text
```

[ ]:
```
combine_df['text'] = combine_df['text'].apply(preprocess_text)
```

[ ]:
```
combine_df
```

[ ]:
```
                                                   text  spam
0     subject natur irresist corpor ident lt realli …     1
1     subject stock trade gunsling fanni merril muzo…     1
2     subject unbeliev new home made easi im want sh…     1
3     subject 4 color print special request addit in…     1
4     subject money get softwar cd softwar compat gr…     1
…                                                   …   …
5165  subject fw crosstex energi driscol ranch 1 3 m…     0
5166  subject put 10 ft transport volum decreas 2500…     0
5167  subject 3 4 2000 follow nom hpl take extra 15 …     0
5169  subject industri worksheet august 2000 activ a…     0
5170  subject import onlin bank alert dear valu citi…     1

[10688 rows x 2 columns]
```

### 0.2.7 Split the data into Train and Test set

[ ]:
```
X = combine_df['text']
y = combine_df['spam']
```

[ ]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
 ↪random_state=42)
```

### 0.2.8 Data Encoding

[ ]:
```
# Convert the data into vectors
vectorizer = CountVectorizer()
X_train_transform = vectorizer.fit_transform(X_train)
X_test_transform = vectorizer.transform(X_test)
```

[ ]:
```
X_train_transform = X_train_transform.toarray()
```

[ ]:
```
X_train_transform.shape,y_train.shape,
```

[ ]:
```
((7481, 49571), (7481,))
```

### 0.2.9 Training Model

[ ]:
```
clf1 = LogisticRegression(max_iter=1000)
clf2 = RandomForestClassifier()
clf3 = SGDClassifier(max_iter=1000,loss='hinge')
clf4 = GaussianNB()
clf5 = MultinomialNB()
```

[ ]:
```
clf1.fit(X_train_transform,y_train)
```

[ ]:
```
LogisticRegression(max_iter=1000)
```

[ ]:
```
clf2.fit(X_train_transform,y_train)
```

[ ]:
```
RandomForestClassifier()
```

[ ]:
```
clf3.fit(X_train_transform,y_train)
```

[ ]:
```
SGDClassifier()
```

[ ]:
```
clf4.fit(X_train_transform,y_train)
clf5.fit(X_train_transform,y_train)
```

[ ]:
```
MultinomialNB()
```

### 0.2.10 Model Prediction on test data

[ ]:
```
clf1_pred = clf1.predict(X_test_transform)
clf2_pred = clf2.predict(X_test_transform)
clf3_pred = clf3.predict(X_test_transform)
```

[ ]:
```
clf4_pred = clf4.predict(X_test_transform.toarray())
```

[ ]:
```
clf5_pred = clf5.predict(X_test_transform)
```

### 0.2.11 Model Evaluation

[ ]:
```
print("Accuracy Score Clf1:",accuracy_score(clf1_pred,y_test))
print("Accuracy Score Clf4:",accuracy_score(clf2_pred,y_test))
print("Accuracy Score Clf6:",accuracy_score(clf3_pred,y_test))
print("Accuracy Score Clf6:",accuracy_score(clf4_pred,y_test))
print("Accuracy Score Clf6:",accuracy_score(clf5_pred,y_test))
```

```
Accuracy Score Clf1: 0.9856563766760212
Accuracy Score Clf4: 0.9766136576239476
Accuracy Score Clf6: 0.9787963829123791
Accuracy Score Clf6: 0.9513564078578111
Accuracy Score Clf6: 0.9840972871842844
```

[ ]:
```
print("Accuracy Score Clf1:",confusion_matrix(clf1_pred,y_test))
print("Accuracy Score Clf4:",confusion_matrix(clf2_pred,y_test))
print("Accuracy Score Clf6:",confusion_matrix(clf3_pred,y_test))
print("Accuracy Score Clf6:",confusion_matrix(clf4_pred,y_test))
print("Accuracy Score Clf6:",confusion_matrix(clf5_pred,y_test))
```

```
Accuracy Score Clf1: [[2351   21]
 [  25  810]]
Accuracy Score Clf4: [[2342   41]
 [  34  790]]
Accuracy Score Clf6: [[2346   38]
 [  30  793]]
Accuracy Score Clf6: [[2349  129]
 [  27  702]]
Accuracy Score Clf6: [[2350   25]
 [  26  806]]
```

[ ]:
```
print("Accuracy Score Clf1:",classification_report(clf1_pred,y_test))
print("Accuracy Score Clf4:",classification_report(clf2_pred,y_test))
print("Accuracy Score Clf6:",classification_report(clf3_pred,y_test))
print("Accuracy Score Clf6:",classification_report(clf4_pred,y_test))
print("Accuracy Score Clf6:",classification_report(clf5_pred,y_test))
```

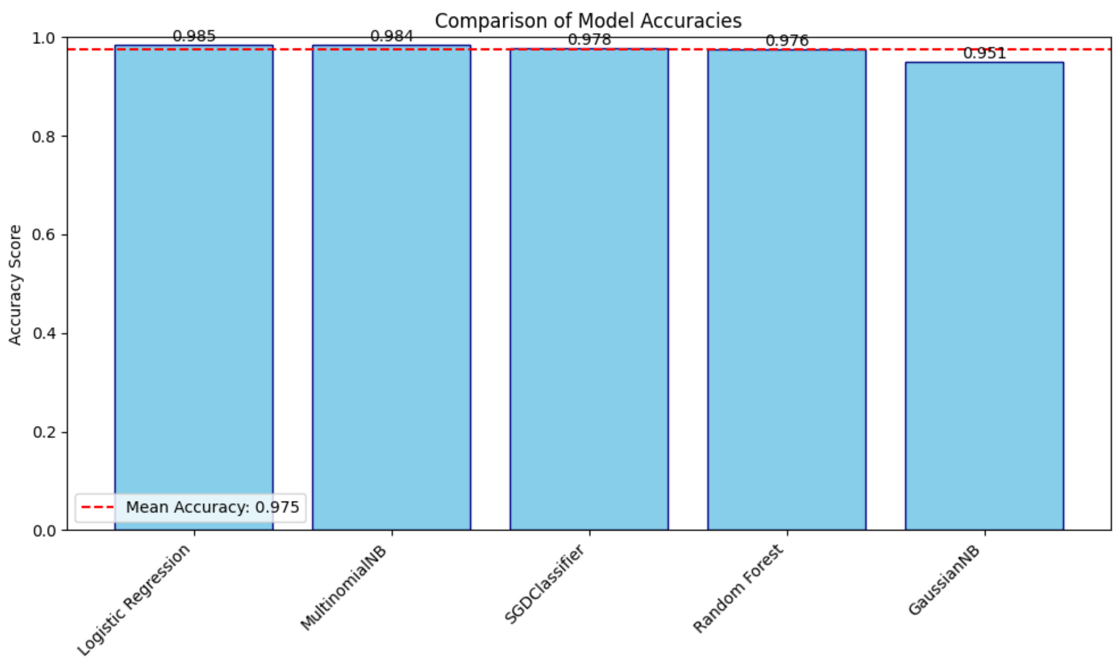
```
Accuracy Score Clf1:               precision    recall  f1-score   support
```

```
           0       0.99      0.99      0.99      2372
           1       0.97      0.97      0.97       835

    accuracy                           0.99      3207
   macro avg       0.98      0.98      0.98      3207
weighted avg       0.99      0.99      0.99      3207

Accuracy Score Clf4:              precision    recall  f1-score   support

           0       0.99      0.98      0.98      2383
           1       0.95      0.96      0.95       824

    accuracy                           0.98      3207
   macro avg       0.97      0.97      0.97      3207
weighted avg       0.98      0.98      0.98      3207

Accuracy Score Clf6:              precision    recall  f1-score   support

           0       0.99      0.98      0.99      2384
           1       0.95      0.96      0.96       823

    accuracy                           0.98      3207
   macro avg       0.97      0.97      0.97      3207
weighted avg       0.98      0.98      0.98      3207

Accuracy Score Clf6:              precision    recall  f1-score   support

           0       0.99      0.95      0.97      2478
           1       0.84      0.96      0.90       729

    accuracy                           0.95      3207
   macro avg       0.92      0.96      0.93      3207
weighted avg       0.96      0.95      0.95      3207

Accuracy Score Clf6:              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2375
           1       0.97      0.97      0.97       832

    accuracy                           0.98      3207
   macro avg       0.98      0.98      0.98      3207
weighted avg       0.98      0.98      0.98      3207
```

### 0.2.12 Visualizing Accuracies of Models

[ ]:
```python
import matplotlib.pyplot as plt
import numpy as np

def plot_model_accuracies(models, scores):
    # Ensure models and scores are lists
    if not isinstance(models, list) or not isinstance(scores, list):
        raise ValueError("Both models and scores should be lists")

    # Ensure the lengths match
    if len(models) != len(scores):
        raise ValueError("The number of models and scores should be the same")

    # Create the plot
    fig, ax = plt.subplots(figsize=(10, 6))

    # Create the bars
    bars = ax.bar(models, scores, color='skyblue', edgecolor='navy')

    # Customize the plot
    ax.set_ylabel('Accuracy Score')
    ax.set_title('Comparison of Model Accuracies')
    ax.set_ylim(0, 1)  # Assuming accuracy scores are between 0 and 1

    # Add value labels on the bars
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.3f}',
                ha='center', va='bottom')

    # Add a horizontal line for the mean accuracy
    mean_accuracy = np.mean(scores)
    ax.axhline(y=mean_accuracy, color='red', linestyle='--', label=f'Mean␣
 ↪Accuracy: {mean_accuracy:.3f}')

    # Rotate x-axis labels for better readability
    plt.xticks(rotation=45, ha='right')

    # Add legend
    plt.legend()

    # Adjust layout and display the plot
    plt.tight_layout()
    plt.show()
```

```
# Example usage:
models = ['Logistic Regression', 'MultinomialNB','SGDClassifier','Random␣
↪Forest','GaussianNB']
accuracy_scores = [0.985, 0.984, 0.978, 0.976, 0.951]  # Replace with your␣
↪actual accuracy scores

plot_model_accuracies(models, accuracy_scores)
```




### 0.2.13 Save the trained models into file

```
[ ]: import pickle
models = [clf1,clf2,clf3,clf4,clf5]
for model in models:
    with open(f"{model}.pkl",'wb') as file:
        pickle.dump(model,file)
```

### 0.2.14 Conclusion:

The `Logistic Regression` model demonstrated the highest accuracy of `98.5%` in classifying emails as spam or non-spam. This project effectively illustrates the process of building and evaluating a text classification model for email spam detection.